FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Weaki - A weakly-typed wiki for incremental software knowledge acquisition

Manuel António Gomes Pereira

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Nuno Flores

July 25, 2016

Weaki - A weakly-typed wiki for incremental software knowledge acquisition

Manuel António Gomes Pereira

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof[a]. Ana Paula Rocha, PhD.

External Examiner: Prof[o]. Luís Reis, PhD.

Supervisor: Prof[o]. Nuno Flores, PhD.

July 25, 2016

Abstract

Software development can be seen as the final stage in a process of knowledge acquisition, during which the acquired knowledge accumulates and evolves towards a source code that fulfills a set of conditions previously defined by the stakeholders. The knowledge acquired this way is stored in the form of software artifacts which have two different aspects: structure and content, that are very difficult to change separately from each other.

To manage the acquired knowledge and the complexity of the acquisition process there are methods, practices and tools that are commonly used to facilitate the acquisition and processing of knowledge. Among these tools, one of the most popular are the wikis.

Wikis provide a good platform to manage knowledge while also supporting collaborative work. However beyond what wikis already provide, it would be useful if they had other aspects such as the recognition of knowledge structures and the support for its evolutionary behavior.

The main goal of this work is to develop an extension to an existing wiki, by adding the functionalities to support structure recognition and type inference, with the goal of aiding in the development of software documentation artifacts, allowing for a gradual increase in the formalization of its structure and content. The usefulness of the wiki extension was validated through an empirical experience with students.

Resumo

O desenvolvimento de software pode ser visto como a última etapa de um processo de aquisição de conhecimento, durante o qual o conhecimento adquirido se acumula e evolui para um código fonte que cumpre um conjunto de condições previamente definidas pelos stakeholders. O conhecimento adquirido desta forma é armazenado sob a forma de artefactos de software, artefactos estes que têm dois aspetos diferentes: a estrutura e o conteúdo, que são muito difíceis de alterar em separado um do outro.

Para gerir o conhecimento adquirido e a complexidade do processo de aquisição existem métodos, práticas e ferramentas que são de maneira geral usados para facilitar a aquisição de conhecimentos e o seu processamento. Entre essas ferramentas, um dos mais populares são as wikis.

Wikis proporcionam uma boa plataforma para gerir o conhecimento para além de suportarem trabalho colaborativo. No entanto para além do que as wikis já fornecem, seria útil se estas tivessem suporte para outros aspetos, tais como o reconhecimento de estruturas de conhecimento e suporte para a sua evolução à medida que conhecimento é adquirido e as mudanças que sofre ao longo do tempo.

O principal objetivo deste trabalho é desenvolver uma extensão para um wiki já existente, adicionando funcionalidades para apoiar o reconhecimento da estruturas da informação, assim como na inferência dessas estruturas pelo conteúdo, com o objetivo de auxiliar no desenvolvimento de artefactos de documentação de software, ao permitir um aumento gradual na formalização da sua estrutura e do seu conteúdo. A utilidade da extensão da wiki foi validada através de uma experiência empírica envolvendo estudantes.

Acknowledgements

First of all to my supervisor Nuno Flores (FEUP) for his guidance and advice, thank you.

To the students that participated in the empirical experiment, thank you for your availability and time.

Last but not least to my family, to my parents for all your support and my brothers for their companionship.

To all of you my sincere thank you.

Manuel Pereira

"We know what we are,
but know not what we can be."

William Shakespeare

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation and Goals . . . 1
- 1.3 Project . . . 2
- 1.4 Structure of the Dissertation . . . 2

2 Bibliographic Review **3**
- 2.1 Knowledge Acquisition in Software Development . . . 3
  - 2.1.1 Knowledge Acquisition Issues . . . 3
- 2.2 Wiki as a tool for Software Documentation . . . 4
  - 2.2.1 What is a Wiki . . . 4
  - 2.2.2 DokuWiki . . . 5
- 2.3 Software Documentation Contents and Structure as Knowledge Evolves . . . 5
- 2.4 Algorithms . . . 5
  - 2.4.1 Tree Edit Distance . . . 6
  - 2.4.2 Tag Cloud . . . 6
- 2.5 Chapter Summary . . . 7

3 TypeInference **9**
- 3.1 Implementing Type Inference . . . 10
  - 3.1.1 Structural Analysis . . . 10
  - 3.1.2 Semantic Analysis . . . 11
- 3.2 Content Assist . . . 12

4 Validation **13**
- 4.1 Empirical Experiment . . . 13
  - 4.1.1 Target Population . . . 13
  - 4.1.2 Test Description . . . 13
  - 4.1.3 Questionnaire . . . 14
- 4.2 Results . . . 14
  - 4.2.1 Background . . . 14
  - 4.2.2 Overall Satisfaction . . . 15
  - 4.2.3 Plug-in . . . 15
  - 4.2.4 Opinion . . . 16
  - 4.2.5 Time Measurement . . . 16
  - 4.2.6 Conclusions . . . 17

5 Conclusions and Future Work **19**
5.1 Contributions 19
5.2 Future Work 19

References **21**

A Test Script **23**
A.1 Test Script A 23
A.1.1 First Part 23
A.1.2 Second Part 23
A.2 Test Script B 24
A.2.1 First Part 24
A.2.2 Second Part 25
A.3 Document 25
A.4 TypeInference 26
A.4.1 Description 26

B Questionnaire **27**

C Template **31**

List of Figures

2.1 Ted normalizing formula 6
2.2 Example of a tag cloud 7

3.1 TypeInference Diagram 9
3.2 On the left the rendering of the wiki text on the right 10
3.3 Wiki page (left) and corresponding tree (right) 11
3.4 Semantic similarity formula 11
3.5 Content Assist displaying type information and template sections 12

4.1 Experiment Step-by-Step 14
4.2 Background Answers Pie Chart 14

LIST OF FIGURES

List of Tables

4.1 Overall Satisfaction Answers 15
4.2 Overall Plug-in Answers 15
4.3 Time Measurement Results 16

Abbreviations

wiki	WikiWikiWeb
TED	Tree Edit Distance
APTED	All Path Tree Edit Distance
RTED	Robust Tree Edit Distance
LRH	Left-Right-Heavy
HTML	HyperText Markup Language
PHP	Hypertext Preprocessor
CSS	Cascading Style Sheets

Chapter 1

Introduction

This introductory chapter provides the context and motivation for this thesis work. The proposed solution and the structure of the document are also described.

1.1 Context

Software development is a knowledge intensive activity and it can be difficult to manage all this knowledge efficiently. The knowledge of software development experts can be vast, most of this knowledge is tacit knowledge which can be very difficult if not impossible to fully materialize in an artifact. While there are countless types of software artifacts most of the time having to pick a specific structure before beginning to write down knowledge can have a negative impact on the capacity of the expert in expressing the knowledge that he has in his mind, having no structure whatsoever can be as negative. Another relevant aspect of software development is collaborative work, most of the time several people work in the same document, simultaneously or not, so the use of a tool that supports collaborative work can be a good step towards streamlining the software development process.

While there are several tools that help with software development and knowledge acquisition, these tools either use a very specific structure for their artifacts or no structure at all. An example of such a tool is a wiki, there are several implementations of wikis all of them with their advantages but lacking when it comes to support for knowledge and artifact evolution.

1.2 Motivation and Goals

Most of the time expressing the tacit knowledge that is the mind of a software developer in a software documentation artifact is a long and complicated task, and when the task is finished the results are still not quite what was expected. This can happen due to several reasons, some of them being inter aspects such as the mood of the software developer or external such as tools

available or the excessive formalizations of the artifacts structure that doesn't allow the knowledge to be expressed conveniently. While a fixed document structure has advantages such a setting a standard from which one can start sometimes this structure can be excessive and restrain the creative mind of the software developer, however the total lack of structure can be negative as well by not setting a starting point. The absence and excess of structure and formalization in software documentation can both have negative impact in knowledge acquisition and therefore a negative impact in software development.

The project goal is to create a tool that allows the user to have a more straightforward approach to software documentation development, focusing more on knowledge acquisition and less on the document structure and formalization.

1.3 Project

The proposed solution consists in an extension to the existing DokuWiki. This extension will add support for weakly typed pages where a wiki page will be classified in one, several or none types, depending on a percentage of similarity with predefined templates. These templates can be changed at any time and evolve as the software documentation evolves, as the templates change so does the types of other wikipages change accordingly, these changes allow users to go back to these pages and choose to update them or to let them stay as they are, depending on what suits the situation best. To help with the creation of new pages as soon as at least one template has relevant similarity with the page that is being worked on it is possible by using the extension to add new sections to the page from the selected template.

1.4 Structure of the Dissertation

In addition to this introductory chapter this Dissertation has the following structure. In Chapter 2 is presented the bibliographic review describing the knowledge acquisition domain as well as its issues. Chapter 2 also a brief overview of wikis as a tool for software documentation and in particular the chosen DokuWiki. It is also described in chapter two several algorithms used in the project development.

In Chapter 3 the proposed solution, the plug-in TypeInference, is described. TypeInference can be divided into two major parts, the analysis and the content assist, the development process of each component and the algorithms used in the analysis are described in detail.

In Chapter 4 is called Validation and describes the empirical experiment where several students tested the TypeInference plug-in and presents the analysis of the results obtained.

The conclusions of this Dissertation are presented in Chapter 5. This final chapter also contains further directions in how to extend the work done in this Dissertation. The plug-in developed can be easily split into its two major parts and therefore operate with different modules. This way different interfaces for the content assist can be used or different algorithms can be used in the analysis.

Chapter 2

Bibliographic Review

In this chapter will be presented the knowledge acquisition domain and model as well as algorithms and other tools relevant for this work.

2.1 Knowledge Acquisition in Software Development

Knowledge acquisition consists of the extraction, structuring and organization of expert knowledge in a way that the knowledge necessary to solve the problem can be captured and stored in a format that can be interpreted by a computer. Knowledge captured this way is the base for the reasoning of an expert system and has as main concerns the appropriate involvement of experts, adequate knowledge elicitation techniques and a structured approach to knowledge acquisition[FFCA09]. Software development can be improved by correctly recognizing the appropriate knowledge structures or representations and having the proper tools to manage them[Rob99].

2.1.1 Knowledge Acquisition Issues

Knowledge acquisition is a complex process with several issues already identified in the literature:

- A large amount of knowledge is in the head of the experts, the capture and share of this knowledge increases its already high value, although this knowledge should be shared in a way that everyone can understand.
- Experts have vast amounts of knowledge, it is important however to focus on the knowledge that is relevant to the issue at hand.
- Each expert doesn't know everything, knowledge should be captured from several experts and the experts should be allowed to interact.

- Experts have a lot of tacit knowledge, this means that experts know more than they can usually account for, however tacit knowledge is hard to describe and nearly impossible to capture.
- Experts are busy people, therefore knowledge elicitation techniques should take the least time possible and if possible be integrated in the work environment.

Knowledge acquisition is a complex and time-consuming process, and usually cause for delays in system development. These delays can and should be mitigated as much as possible through the use of appropriate methods and techniques[FFCA09, Lio92].

2.2 Wiki as a tool for Software Documentation

Software documentation plays a central role in many software development tasks, however its importance is not always recognized. Most of the development effort is spent formalizing information, reading and understanding requirements, specifications and other documents with the intention of producing source code. Good software documentation usually provides multiple views of a system with different levels of abstraction, and depending on the concrete aspect of the document it may include several notations from different types of artifacts. This level of complexity leads to software documentation being a hard and costly task to perform, especially when not using any adequate tool or method[AD05].

Software artifacts are products of software development, the things that developers work with. They may be part of a set of deliverables or may describe or support the process of developing software and impact the creation of other software artifacts[Cor15].

Several tools are used to manage and develop software artifacts as knowledge increments and evolves, one such tool is the wiki, wikis are nowadays massively used by developers due to the benefits they provide in knowledge management and collaborative authoring. Wikis have been found to be especially useful in agile environments and in the creation of lightweight documentation[Cor15].

2.2.1 What is a Wiki

A WikiWikiWeb, or simply a Wiki, is a collaborative tool capable of presenting and editing online information using a simple web browser[Agu03, AD05]. The first wiki was created by Ward Cunningham 1995[Cun] and since then several other implementations were made based on the same design principles. These principles are:

- Open - any user can edit a page.
- Incremental - pages can cite other pages even if they are yet to be created.
- Organic – the structure and contents of the site are open to edition evolution.
- Mundane – simple to use with small number of text conventions for formatting.

- Universal – the mechanisms of editing and organizing are the same so that any writer is automatically an editor and organizer.
- Overt – The formatted and printed out-put will suggest the input required to reproduce it.
- Unified – Page names will be drawn from a flat space so that no additional context is needed to interpret them.
- Precise – pages will be titled to avoid name clashes, typically with noun phrases.
- Tolerant – Interpretable behavior is preferred to error messages.
- Observable – Activity with the site can be reviewed by any other visitor .
- Convergent - Duplication can be removed by finding and citing related content[Cor15, Cun].

2.2.2 DokuWiki

DokuWiki is an example of such an implementation. Being simple to use and lightweight, DokuWiki has a clean and readable syntax and doesn't require a database. The built in access controls and authentication make it a especially useful wiki in enterprise context and the large amount of plug-ins developed by the community allow for a broader range of uses beyond that of a traditional wiki[Goh].

While there are other wiki implementations DokuWiki was chosen for this work not only because of the above mentioned factors but also because of the familiarity with the wiki and therefore an increased productivity. For more detailed information refer to Wiki Matrix website[wik].

2.3 Software Documentation Contents and Structure as Knowledge Evolves

Software systems can hardly be seen as immutable. The knowledge of project stakeholders about a system under construction is always bound to change as time progresses. This implies that software artifacts are bound to change as they may need to be adapted to new understandings. Although as knowledge evolves artifacts may not be easy to adapt accordingly especially if a structural change is needed[Cor15]. The interdependence in between artifacts or parts of artifacts suggests that changing a particular artifact can impact the other artifacts at the same or even other levels of abstraction. To help with consistency of the documentation refactoring tools use propagation mechanisms to allow changes to a specific artifact to be reflected in all the documentation[Cor15].

2.4 Algorithms

The following algorithms will be used to assess the level of similarity between wiki pages. Both provide different approaches that allow to review both structure and semantic content of the pages in order to achieve the most accurate degree of similarity.

2.4.1 Tree Edit Distance

The standard approach to tree similarity is to calculate the tree edit distance, that is to calculate the minimum number of node operations (insert, edit and delete) that one tree has to undergo to become the other[PA16, But04]. There are several implementations of TED algorithms such as RTED (Robust Tree Edit Distance)[PA15], and APTED(All Path Tree Edit Distance)[PA16]. RTED implements LRH (left-right-heavy) strategy[PA15] however this approach trades off memory efficiency for computational time, APTED however uses an All-Path approach that is at least just as fast as RTED without trading off memory efficiency[PA16]. The information obtained from this algorithm can be converted into a similarity metric by normalizing the number of edit operations with the number of nodes from the largest tree as can be seen in the following formula. Let Di and Dj be documents and Ni,Nj the number of nodes from the corresponding trees.

$$TED(D_i, D_j) = \frac{editDistance(D_i, D_j)}{max(|N_i|, |N_j|)}$$

Figure 2.1: Ted normalizing formula

2.4.2 Tag Cloud

A tag cloud is usually seen as visual representation of a set of words, where words with more weight are presented in a larger font, however a tag cloud can also be seen as a list of weighted words[SCH08, GXF08]. Such a weighted list can be used to compare two document by overlapping their corresponding weighted lists and checking how many tags match and their relative position, however this technique while simple by itself it's not satisfactory, the major issue is that pages that follow the same schema, an HTML document for example, two documents can have very similar contents but one page contains a large number of a certain tag while the other only has a few occurrences. To compensate for this a weighted tag system can be used, but in documents types such as HTML where the tag set is limited but the tag frequency varies significantly this metric will have poor results[But04].

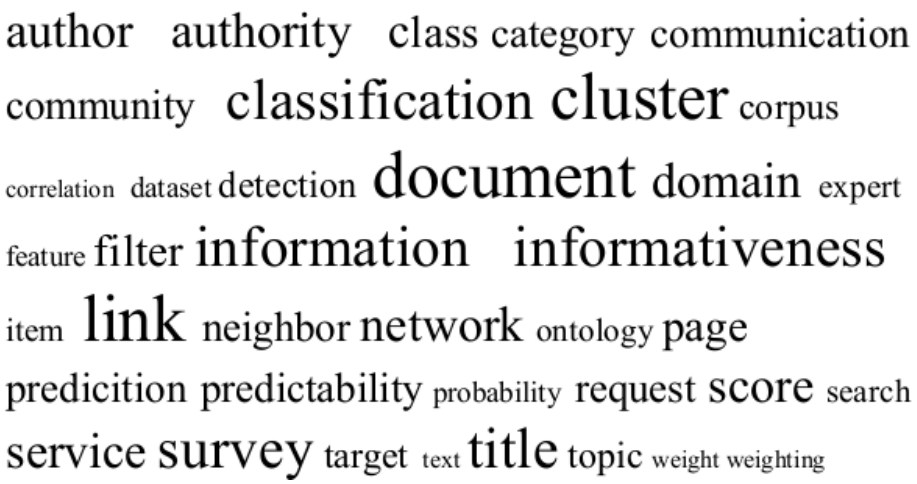


Figure 2.2: Example of a tag cloud

2.5 Chapter Summary

In this chapter it was presented the knowledge acquisition domain and model. There was also a overview of wikis as a tool for software documentation and a quick presentation of DokuWiki, its advantages and why it was chosen for this thesis work. There was also a description of the algorithms used in the TypeInference plug-in. In the next chapter the implementation will be described in detail.

Chapter 3

TypeInference

TypeInference is an extension for DokuWiki. It can be divided in two distinct aspects, one is type inference itself where the wiki pages are analyzed and compared to the templates and another is the content assist where the user is provided with the type inference information and menus to select sections from the templates that were found to be relevant.

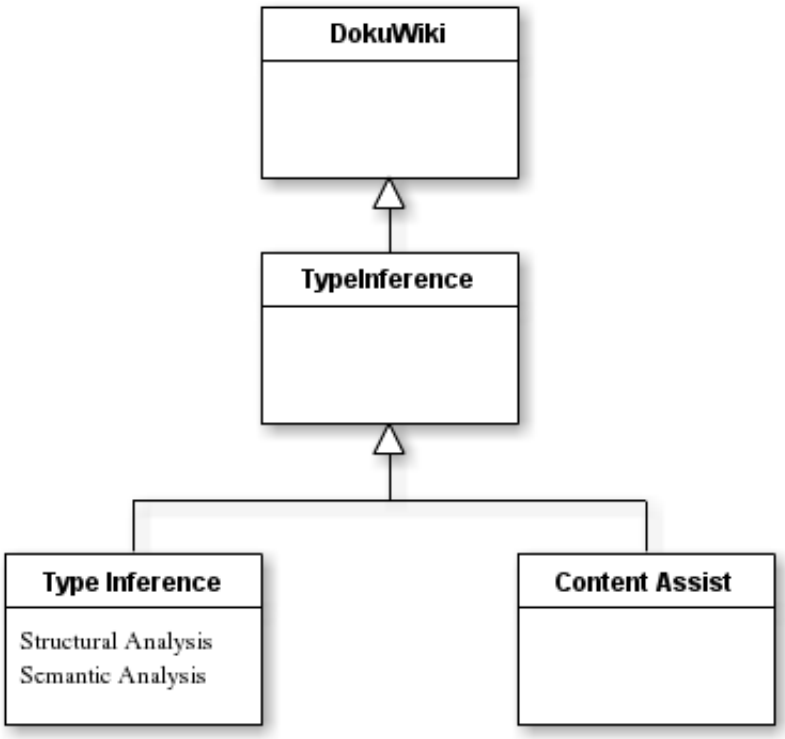


Figure 3.1: TypeInference Diagram

3.1 Implementing Type Inference

DokuWiki supports several types of plug-ins, such as syntax, action, auth and others[1]. TypeInference is an action plug-in. Action plug-ins in DokuWiki are triggered by events that occur in the wiki, TypeInference registers two hooks for events, one in order to access the text of a wiki page when the page is saved and another to load information onto the JavaScript variable `$JSINFO`.

The type inference component is responsible for the analysis and matching of the wiki pages with the templates. These templates follow a specific structure as can be seen in the following image where can be seen the wiki text and its visual rendering. The analysis is done in two phases, on the first phase the wiki pages are compared at the structural level and on the second phase they are compared at the semantic level. The type inference match is the average of the values obtained on these two phases. Both these analyses are implemented in PHP. The wiki text is obtained using the DokuWiki function `rawWiki($id)` and parsed with regular expressions using the PHP function `preg_match()`.

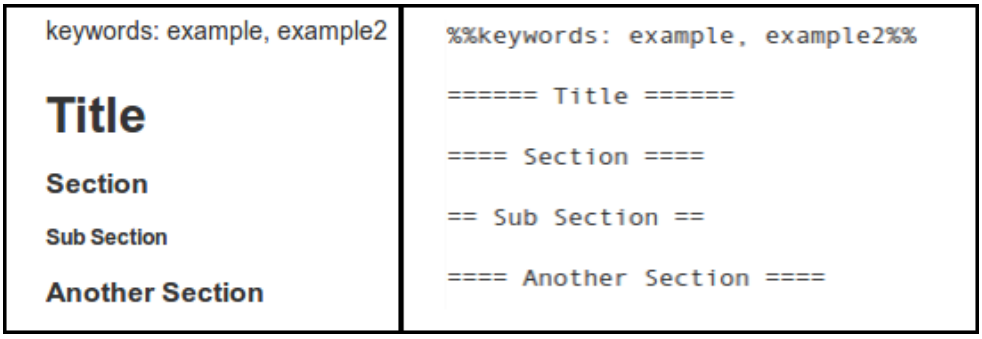


Figure 3.2: On the left the rendering of the wiki text on the right

3.1.1 Structural Analysis

The structural similarity of the documents is measured with the use of a TED based metric where both documents are converted into trees and the edit distance is calculated with the use of an APTED implementation[PA, PA15, PA16], this edit distance is normalized by the formula presented in Chapter 2 and converted to a percentage value based on the fact that zero edit distance corresponds to a hundred percent match and the largest number of nodes of the largest tree corresponds to a zero percent match.

The documents trees are created by using `explode()` to split the document into lines, this lines are filtered into section titles and placed in the appropriate level of the tree after cleaning the wiki syntax characters. The APTED implementation is a java implementation stored in a jar file and is called using the function `exec()`.

[1] https://www.dokuwiki.org/plugins

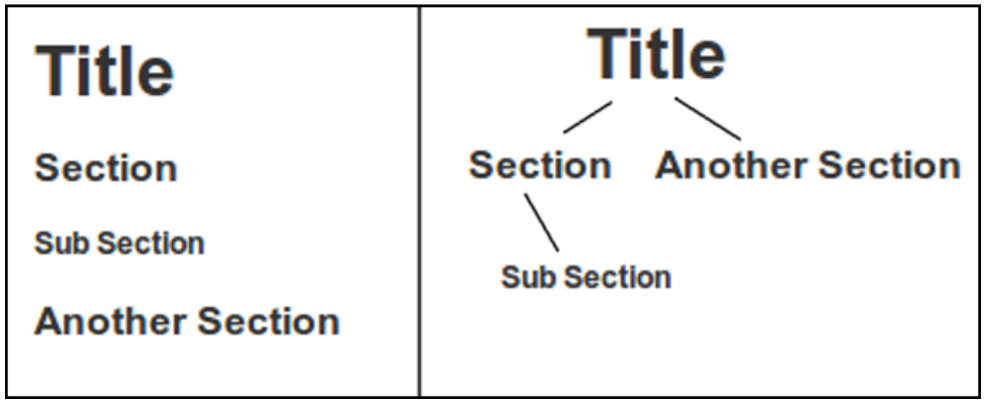


Figure 3.3: Wiki page (left) and corresponding tree (right)

3.1.2 Semantic Analysis

The semantic analysis is done by creating a weighted list based on the count of each word used on the wiki page. From this weighted list a set of "stop-words"[Stp] are removed, stop words are words that aren't important to the semantic context of the text and words that only contribute to the text syntax. At this point the weighted list is checked against the templates keywords in order to check how many words are present, at this point zero words found means a zero percent match, while any other amount of words will directly affect the result percentage. The average position of the found words on the weighted list is divided by the average difference between the highest N words and lowest N words (where N is the number of found words) on the weighted list, meaning that if the highest N words on the list are the found words then it's a hundred percent match and if the number of found words is the lowest N words then it's a zero percent match. This value is then multiplied by the value obtained when checking how many keywords could be found in the weighted list giving the final value as can be seen in the following formula, where K is the total number of keywords, A is the average position of the found words, AM is the average position of the highest words on the weighted list and Am the average position of the lowest words on the weighted list. After the match value is obtained, the average between both analyses is calculated and the information stored in the variable `$JSINFO`, so that it is available to the JavaScript component of the plug-in.

$$R = \frac{N}{K} * \frac{A}{\frac{AM - Am}{100}} * 100$$

Figure 3.4: Semantic similarity formula

3.2 Content Assist

The content assist displays the type inference information on the bottom of the text area, and when the page is in edit mode the information is colored blue and when clicked shows a dropdown menu with a list of sections from the selected template as shown on Figure 3.5, when the user selects a section, that section is automatically added to the bottom of the wiki page currently being edited. The type information will only be shown when there are any relevant matches with the templates and when more than 6 relevant matches occur only the top 6 will be shown.

While the other component of the plug-in is implemented in PHP, the content assist is implemented in JavaScript and uses HTML and CSS to display the type information and user interface. The type information is read from the `$JSINFO` variable and displayed in a div bellow the text area. When the page is in edit mode this div also contains a dropdown menu for each of the matched templates, in this menu each element is a button that appends the corresponding section title to the page text.

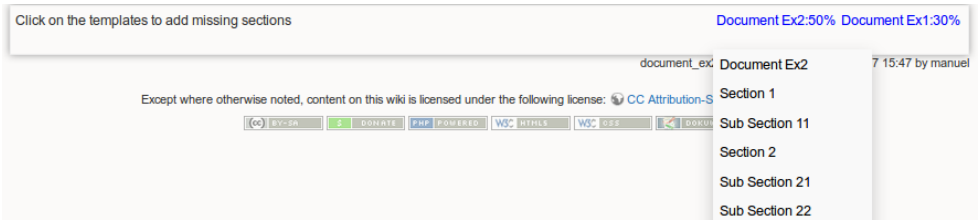


Figure 3.5: Content Assist displaying type information and template sections

Chapter 4

Validation

In this chapter the empirical experiment is going to be explained in detail, starting with the target population, test description and metrics used. Following, the test results are described.

4.1 Empirical Experiment

The objective of this experiment was to evaluate the plug-in. Several aspects were evaluated such as usefulness, user interface and possible time gains when using the plug-in.

4.1.1 Target Population

The target population of this experiment were students of the Integrated Master in Informatics and Computing Engineering, lectured at University of Porto, Faculty of Engineering. By selecting students from this Master's degree is expected that their experience with developing software documentation will allow them to overcome most of the issues related to software documentation that are raised during the experiment.

4.1.2 Test Description

The test consists of two distinct parts, in the first part the test subjects are asked to elaborate a requirements document of a clock [Clo] application (either desktop or mobile) and of a calculator [Cal] (either desktop or mobile) in the second part. The test subjects were divided in two groups with 4 and 5 students each, group A that worked with the TypeInference plug-in in the second part of the test and group B that worked with the plug-in in the first part. The test subjects were provided with the appropriate test script for their group (Appendix A) which in addition to information about the tasks also provided a brief description of the TypeInference plug-in, a suggestion of sections for the requirements document and a link to the DokuWiki syntax.

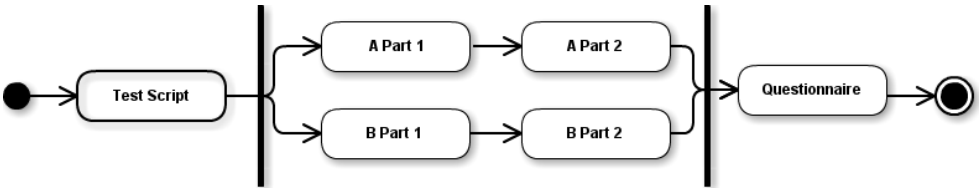


Figure 4.1: Experiment Step-by-Step

4.1.3 Questionnaire

The questionnaire (Appendix B) was design with yes or no questions and statements where the user can pick a number from a scale in the following format: 1 (disagree), 2 (somewhat disagree), 3 (neither agree nor disagree), 4 (somewhat agree) and 5 (agree). These questions and statements were grouped in the following groups:

1. **Background.** This group of questions aims to evaluate the subjects background with wiki and if it may affect the test results.

2. **Overall Satisfaction.** This group aims to dismiss some validation threats such as high pressure environment or bad test environment.

3. **Plug-in.** This group aims to evaluate the overall performance and usefulness of the plug-in.

4. **Opinion.** In this question the test subjects are expected to give their general opinion about the plug-in and any suggestions they may have to improve it.

4.2 Results

In this section the test results and the Questionnaire will be presented and discussed.

4.2.1 Background

From these questions is possible to assess that the test subjects were mostly familiar with DokuWiki and wikis in general.

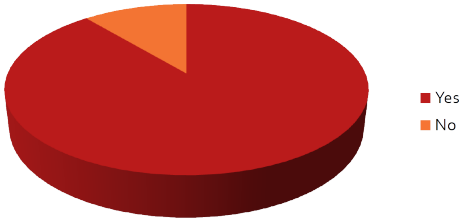


Figure 4.2: Background Answers Pie Chart

4.2.2 Overall Satisfaction

The results from this section show that the subjects could execute the test comfortably and without major effect from external factors. However group A was slightly less affected by these factors.

Statements	Group A	Group B	Total
OS1	4.5	4.2	4.34
OS2	4.75	4	4.34
OS3	4.75	4	4.34

Table 4.1: Overall Satisfaction Answers

OS1. I found the test environment conditions were acceptable.

There is a slight difference in between the two groups but it is not significative.

OS2. The test environment was distraction free.

The test was conducted in several different locations so it is understandable that one of the groups felt more affected by distractions in the test environment.

OS3. I didn't feel pressured while doing the test.

The test subjects were told that they could take as much time as they wanted to execute the given tasks however some subjects might have felt more pressured due to other commitments besides the experiment.

4.2.3 Plug-in

In general the results from this section show that the subjects found the plug-in useful, however the interface aspect was found to be lacking in comparison.

Statements	Group A	Group B	Total
P1	4.75	4.6	4.67
P2	4	4.6	4.34
P3	3.5	3.8	3.67
P4	4.25	4.6	4.45
P5	4.5	4.6	4.56

Table 4.2: Overall Plug-in Answers

P1. I found the TypeInference plug-in useful.

Both groups found the plug-in to be useful without any significative difference.

P2. Using TypeInference allowed me to be less worried about the structure of the document.

The test subjects in group B felt more inclined to rely on the plug-in for the structure of the document, however group A response was also positive.

P3. The TypeInference plug-in information was conveniently displayed on the screen.

Both groups of subjects felt that the interface and the way the information was displayed was somewhat lacking.

P4. I would rather use DokuWiki with the TypeInference plug-in than without.

The test subjects in group B are slightly more inclined to use the plug-in than the subjects in group A, but the overall response is positive.

P5. The TypeInference plug-in can be useful in other knowledge intensive areas besides Software Engineering.

Both groups agree that the plug-in can be useful in other areas of study.

4.2.4 Opinion

In this section almost all test subjects expressed their opinions and suggestions of the plug-in. The general opinion was that the plug-in was useful and allowed the user to quickly structure a document saving time for content production. Most users suggested that the location where the information was displayed should be changed to a more visible one. Another suggestion was that the information should be updated as the user types content and not only when the page is saved or previewed.

4.2.5 Time Measurement

	Group A	**Group B**
Part One	13m04s	10m53s
Part Two	12m29s	10m44s

Table 4.3: Time Measurement Results

Group B subjects were faster executing both tasks, and overall there wasn't much difference in the execution of both tasks by either group. The difference in time between the groups can be due to any given number of reasons such as previous experience in creating software documentation and external factors such as distractions. Overall whether the plug-in can speed the process of software documentation creation remains inconclusive since no appreciable difference could be noted in the execution time between tasks with and without the plug-in.

4.2.6 Conclusions

The experiment took place without any major issues and the test subjects found the environment acceptable.

In general, even though most test subjects found the plug-in interface lacking, the overall response to the plug-in was good and most test subjects found it to be useful when creating software documentation. Whether the plug-in can speed the process of software documentation creation remains inconclusive and more thorough testing is required. Several improvements were suggested by the test subjects such as updating the type information as the user types and placing this information in a more visible location.

Chapter 5

Conclusions and Future Work

Knowledge acquisition is an intensive activity and as such is very time consuming. Software documentation developers use several tools to help manage this process and improve the knowledge acquisition rate. However the available tools are lacking in some aspects, one of the most impactful being support for software documentation artifact evolution and different levels of software documentation artifact structure formalization.

5.1 Contributions

This Dissertation contributes to a solution for the above mentioned issue, the proposed and developed extension allows the software documentation developer to update document templates overtime and gradually increase the formalization of software artifacts structure.

Validation was achieved through an empirical experiment. This experiment concluded that the extension is a useful tool to create software documentation artifacts and that the tool allows the software documentation developer to focus more on the actual contents of the document than on its structure. Whether the extension helps speed up the software documentation development was inconclusive.

5.2 Future Work

This Dissertation is just a starting point, while the TypeInference plug-in can be improved both at the type analysis level and at the interface level, there many more undeveloped functionalities that make the Weaki wiki and without them all together is hard to gauge the tool's true potential.

References

[AD05] Ademar Aguiar and Gabriel David. Wikiwiki weaving heterogeneous software artifacts. In *Proceedings of the 2005 international symposium on Wikis (San Diego, California, USA)*, pages 67–74. ACM, 2005.

[Agu03] Ademar Aguiar. *A minimalist approach to framework documentation*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, September 2003.

[But04] D. Buttler. A short survey of document structure similarity algorithms. In *The 5th International Conference on Internet Computing Las Vegas, NV, United States*, March 2004.

[Cal] Calculator. Available on https://lh6.ggpht.com/942KbwPIon7xQet0Qv5F0Orj70Ob3zlGq48NWbWQgx1RkE7MXJ_5Arz5tEclNiRMwYK3=w300, accessed on 24/6/2016.

[Clo] Clock. Available on https://thecustomizewindows.com/wp-content/uploads/2012/04/Metallic-Flipping-Clock-Widget.jpg, accessed on 24/6/2016.

[Cor15] Filipe F. Correia. *Documenting Software With Adaptive Software Artifacts*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, January 2015.

[Cun] Ward Cunningham. Wiki design principles. Available on http://c2.com/cgi/wiki?WikiDesignPrinciples, accessed on 24/6/2016.

[FFCA09] Nuno Flores Filipe F. Correia, Hugo S. Ferreira and Ademar Aguiar. Incremental knowledge acquisition in software development using a weakly-typed wiki. In *Proceedings of the 5th International Symposium on Wikis and Open Collaboration (Orlando, Florida, USA)*. ACM, October 2009.

[Goh] Andreas Gohr. Dokuwiki. Available on https://www.dokuwiki.org, accessed on 24/6/2016.

[GXF08] Fernando Morgado Geraldo Xexéo and Patrícia Fiuza. Differential tag clouds: Highlighting particular features in documents. *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on*, 3, 2008.

[Lio92] Y. I. Liou. Knowledge acquisition: issues, techniques, and methodology. In *Proceedings of the 1990 ACM SIGBDP conference on Trends and directions in expert systems, Orlando, Florida, United States*, page 212–236. ACM, 1992.

REFERENCES

[PA] M. Pawlik and N. Augsten. Apted. Available on http://tree-edit-distance.dbresearch.uni-salzburg.at/#download, accessed on 24/6/2016.

[PA15] DM. Pawlik and N. Augsten. Tree edit distance: Robust and memory-efficient. *ACM Transactions on Database Systems (TODS)*, 40, March 2015.

[PA16] DM. Pawlik and N. Augsten. Tree edit distance: Robust and memory-efficient. *Information Systems*, 56:157–173, March 2016.

[Rob99] P. N. Robillard. The role of knowledge in software development. In *Communications of the ACM*, volume 42, page 87–92. ACM, January 1999.

[SCH08] James Sinclair and Michael Cardew-Hall. The folksonomy tag cloud: when is it useful? *Journal of Information Science*, 34:15–29, February 2008.

[Stp] Stop-words. Available on http://www.lextek.com/manuals/onix/stopwords1.html, accessed on 24/6/2016.

[wik] Wiki matrix. Available on http://www.wikimatrix.org/, accessed on 24/6/2016.

Appendix A

Test Script

A.1 Test Script A

This test is divided into two parts. In both parts you will be asked to create a requirements document, in one of the parts you will do so in a clean DokuWiki and in a DokuWiki with the TypeInference plug-in. While doing the test some sections may require diagrams, for the purposes of the test you are not required to make them.

A.1.1 First Part

Create a document where you specify the requirements of a simple clock application (it can be either a desktop or mobile application), the document should not exceed 6 requirements. The clock is very simple like the one in the image, it can be digital or analog and have any timing functions you like. Follow this link Document for additional information about the document and this link DokuWiki SyntaxDokuWiki Syntax for information about the DokuWiki syntax. In this first part you will work with a simple DokuWiki.

Start

A.1.2 Second Part

Create a document where you specify the requirements of a simple calculator application (it can be either a desktop or mobile application), the document should not exceed 10 requirements. The calculator is very simple like the one in the image. Follow this link Document for additional

information about the document and this link DokuWiki SyntaxDokuWiki Syntax for information about the DokuWiki syntax. In this second part you will work with a DokuWiki with the TypeInference plug-in.

Start

Please answer this questionnaire when you are finished.

Helpful Links

Document

TypeInference

DokuWiki Syntax

A.2 Test Script B

This test is divided into two parts. In both parts you will be asked to create a requirements document, in one of the parts you will do so in a clean DokuWiki and in a DokuWiki with the TypeInference plug-in. While doing the test some sections may require diagrams, for the purposes of the test you are not required to make them.

A.2.1 First Part

Create a document where you specify the requirements of a simple clock application (it can be either a desktop or mobile application), the document should not exceed 6 requirements. The clock is very simple like the one in the image, it can be digital or analog and have any timing functions you like. Follow this link Document for additional information about the document and this link DokuWiki SyntaxDokuWiki Syntax for information about the DokuWiki syntax. In this first part you will work with a DokuWiki with the TypeInference plug-in.

Start

A.2.2 Second Part

Create a document where you specify the requirements of a simple calculator application (it can be either a desktop or mobile application), the document should not exceed 10 requirements. The calculator is very simple like the one in the image. Follow this link Document for additional information about the document and this link DokuWiki SyntaxDokuWiki Syntax for information about the DokuWiki syntax. In this second part you will work with a simple DokuWiki.

Start

Please answer this questionnaire when you are finished.

Helpful Links

Document
TypeInference
DokuWiki Syntax

A.3 Document

Description

The document should contain the necessary information about the product and its requirements.

Some Possible Sections

Project Description

This section consists in a general description of the project.

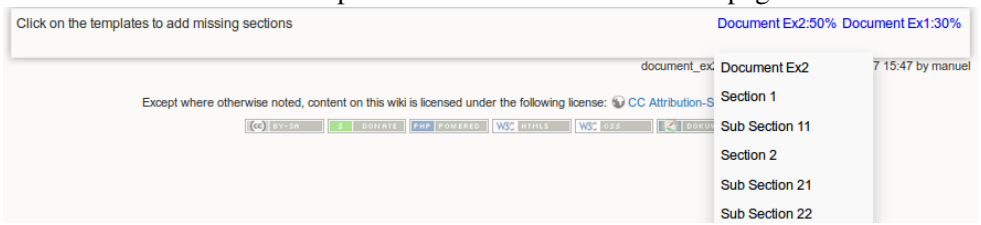
Requirements

This section generally contains a small introductory statement.

A.4 TypeInference

A.4.1 Description

This plugin can be divided into two major parts. The first runs two analysis algorithms through the wiki page text, one reads document structure and the other semantic content, this analysis is matched with several templates, that are saved under a wiki page named typeinference, and shows the highest percent matches under the text area. The second is enabled in the edit mode, and allows the user to click on the name of the templates in the footer and select from a drop-down menu several sections of the template and add them to the current page.



Notes In other for the plugin to be able to compare the document structure the following syntax must be followed

```
====== Title ======

==== Section ====

== Sub Section ==

==== Another Section ====
```

Appendix B

Questionnaire

Bellow is a printed version of HTML questionnaire used to collect the students answers after the test.

Weaki

This form intends to evaluate your previous knowledge about dokuwiki, and gather user feedback about the plug-in TypeInference

*Obrigatório

User Test Code *

Sua resposta

B1. Have you worked with wikis before? *

- ◯ Yes
- ◯ No

B2. Did you already know about DokuWiki before doing this test? *

- ◯ Yes
- ◯ No

B3. Have you worked with DokuWiki before? *

- ◯ Yes
- ◯ No

OS1. I found the test environment conditions were acceptable. *

	1	2	3	4	5	
disagree	◯	◯	◯	◯	◯	agree

OS2. The test environment was distraction free. *

	1	2	3	4	5	
disagree	○	○	○	○	○	agree

OS3. I didn't feel pressured while doing the test. *

	1	2	3	4	5	
disagree	○	○	○	○	○	agree

P1. I found the TypeInference plug-in useful. *

	1	2	3	4	5	
disagree	○	○	○	○	○	agree

P2. Using TypeInference allowed me to be less worried about the structure of the document. *

	1	2	3	4	5	
disagree	○	○	○	○	○	agree

P3. The TypeInference plug-in information was conveniently displayed on the screen. *

	1	2	3	4	5	
disagree	○	○	○	○	○	agree

P4. I would rather use DokuWiki with the TypeInference plug-in than without. *

	1	2	3	4	5	
disagree	○	○	○	○	○	agree

P5. The TypeInference plug-in can be useful in other knowledge intensive areas besides Software Engineering. *

	1	2	3	4	5	
disagree	○	○	○	○	○	agree

O1. Give a brief opinion on what was your experience with the TypeInference plug-in, giving feedback on features, caveats and possible improvements.

Sua resposta

ENVIAR

Nunca envie senhas pelo Formulários Google.

Appendix C

Template

keywords: development, requirements, functionalities, system

Requirements Elicitation Report

Introduction

Project Description

Product Functionalities

Requirements

Functional Requirements

Non Functional Requirements

User Stories

Use Cases Diagrams

Domain Model

Concept Diagram